



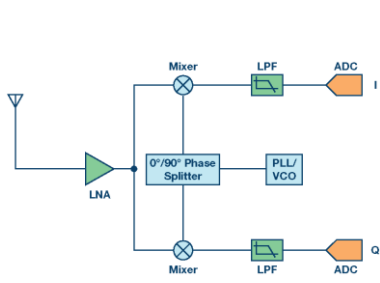
PlutoSDR, the Making of an Ultra Low Cost, High Performance Linux Based Software Defined Radio

**Michael Hennerich
Analog Devices GmbH**

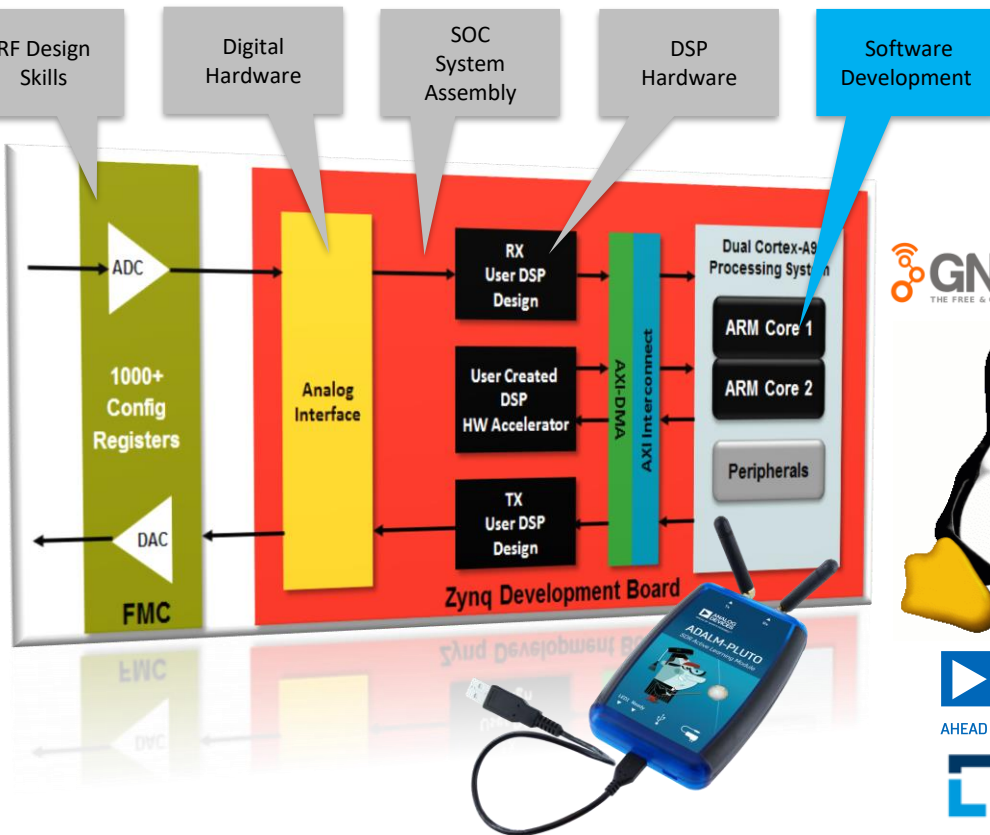
#lfelc



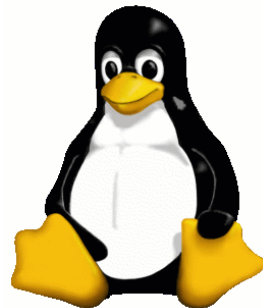
ADALM-PLUTO Active Learning Module (PlutoSDR)



- Introduce the fundamentals of
 - Software-Defined Radio (SDR)
 - Radio Frequency (RF)
 - Wireless communications
 - Embedded Linux
 - FPGA HDL development
 - Open Source Software
 - Open Source Hardware
- ...to everyone
- Designed for users at all levels and all backgrounds
- Student affordable



 **GNU Radio**
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM



 **ANALOG DEVICES**
AHEAD OF WHAT'S POSSIBLE™

 **THE LINUX FOUNDATION**

Requirements



ADALM-PLUTO

SDR Active Learning Module



- Low-cost Hardware
- Zero-cost Ecosystem
- Open
- Reliable
- Fail Safe / Brick-free
- High Performance
- Fast
- Intuitive
- Ease of use
- Supportable
- Extensible
- Flexible
- Cross Platform



AHEAD OF WHAT'S POSSIBLE™

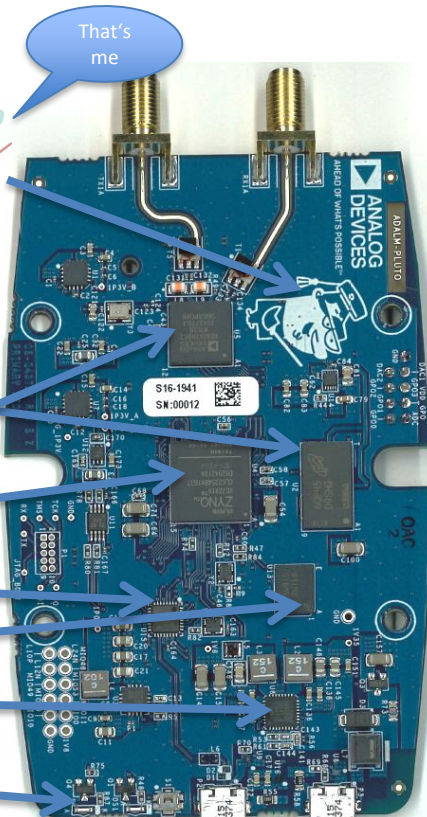
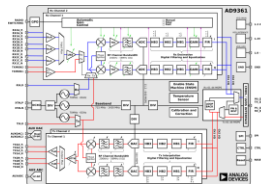
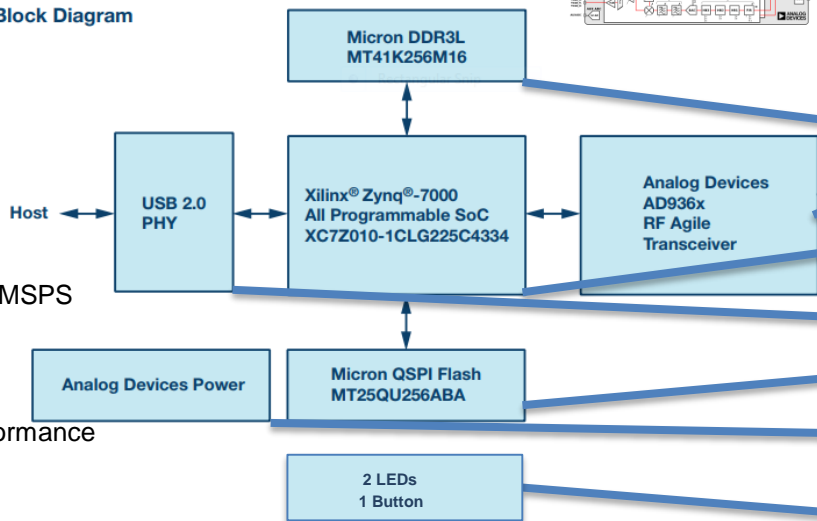


ADALM-PLUTO aka PlutoSDR – What's inside?

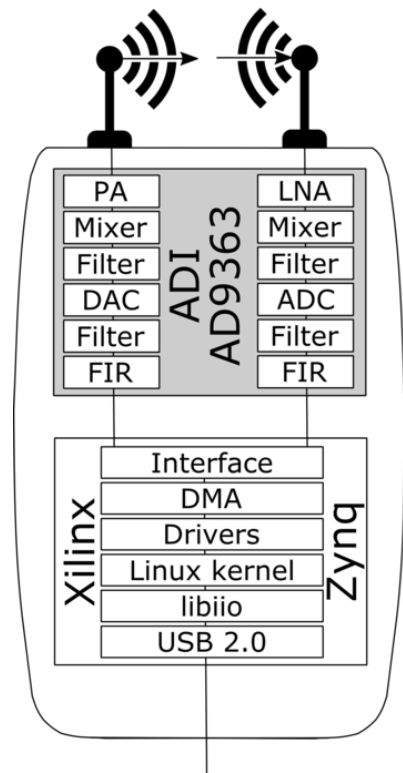


Simplified Block Diagram

- AD9363
- Xilinx Zynq-7010
- 512 MB DDR3
- 32 MB SPI NOR
- USB 2.0 OTG
- Captures I/Q Samples
 - 12-bits
 - 65.1 kSPS to 61.44 MSPS
- Tuning range:
 - ▶ 325 MHz to 3.8 GHz
 - ▶ Guaranteed performance
 - ▶ 70 MHz to 6.0 GHz
 - ▶ Unknown specs



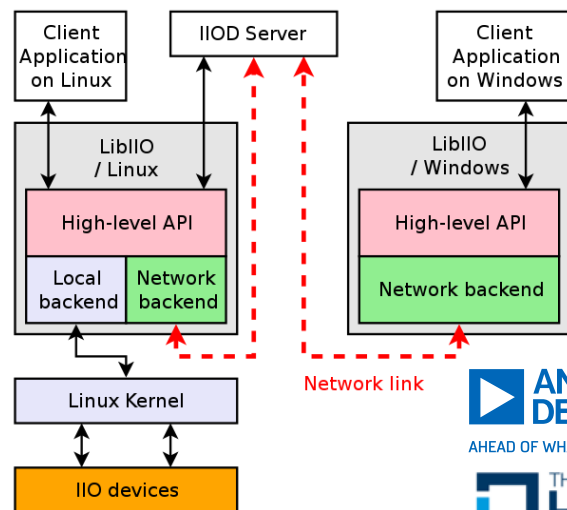
ADALM-PLUTO – Software stack



- Runs Linux inside the device
- Uses Linux's IIO framework to expose I/Q data and control
- Multi-Function USB Device
 - Native IIO over USB (FunctionFS)
 - Serial over USB (CDC ACM)
 - Kernel console
 - COMx, ttyACMx
 - Ethernet over USB (RNDIS)
 - Mass Storage (MSD)
 - Device Firmware Update (DFU)
- USB Host
 - USB devices
 - Ethernet, WIFI, Audio, HID, MSD

• Cross Platform

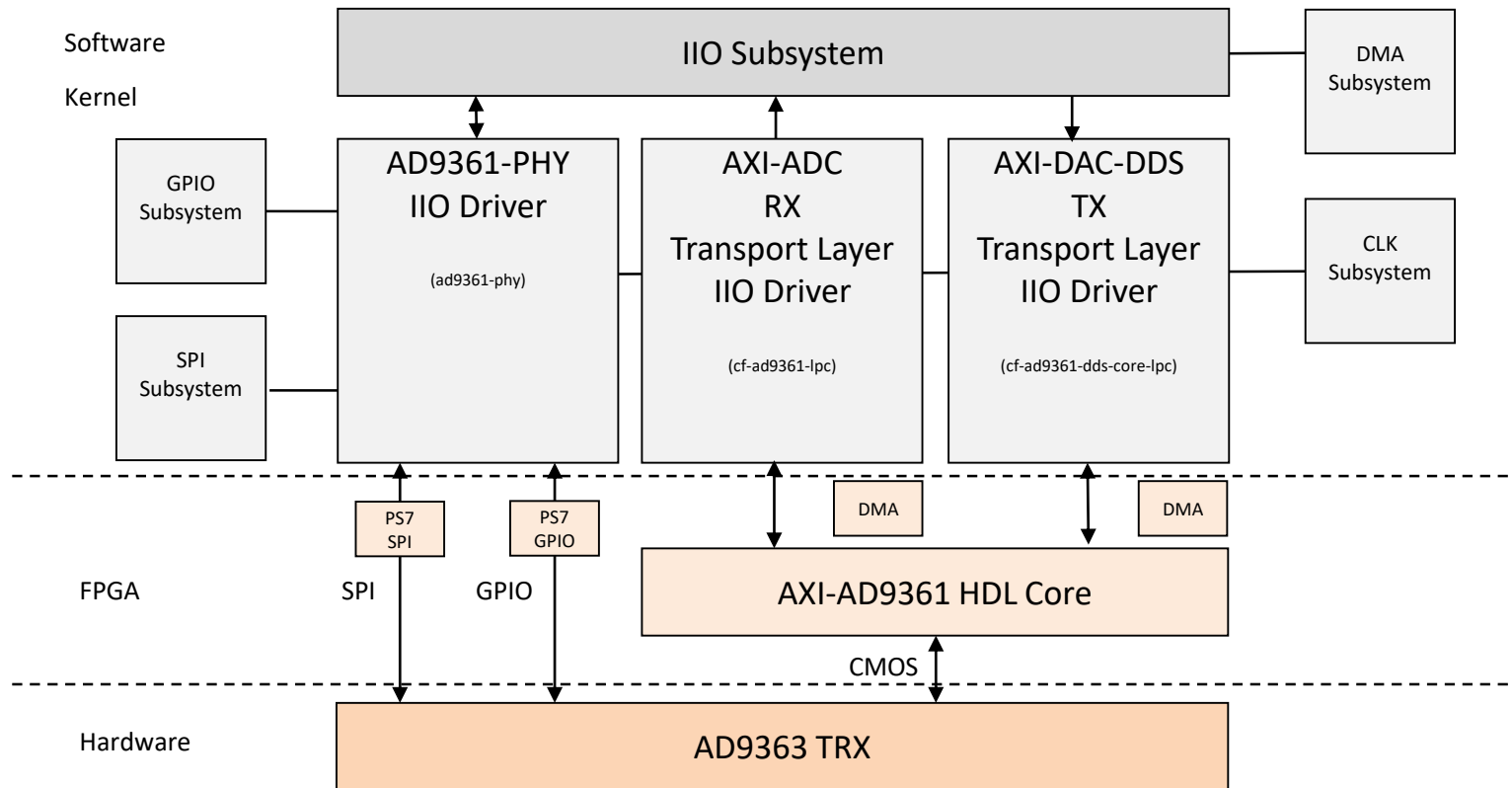
- Windows
- Linux
- MAC



ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

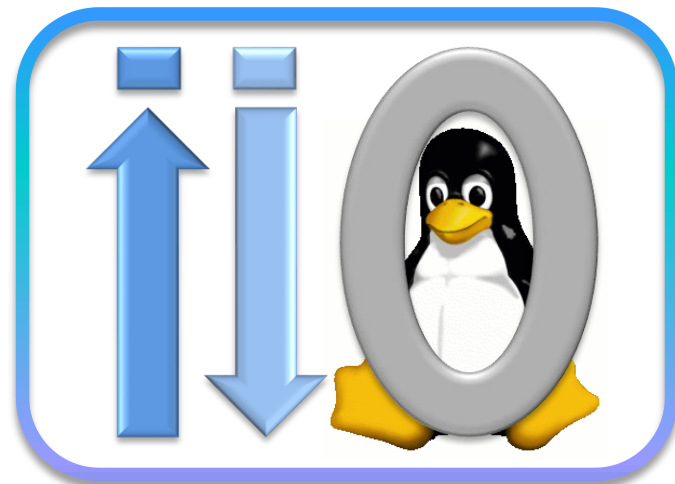
THE LINUX FOUNDATION

Software, Programmable Logic & Hardware



What is IIO?

- Linux kernel **I**ndustrial **I**nput / **O**utput subsystem
- Not really just for Industrial IO
 - All non-HID IO
 - ADC, DAC, light, accelerometer, gyro, magnetometer, humidity, temperature, pressure, rotation, angular momentum, chemical, health, proximity, counters, etc.
- In the upstream Linux kernel for 10 years.
- Mailing list:
 - linux-iio@vger.kernel.org



<https://www.kernel.org/doc/html/latest/driver-api/iio/index.html>


AHEAD OF WHAT'S POSSIBLE™


THE
LINUX
FOUNDATION

Why use IIO for SDR?

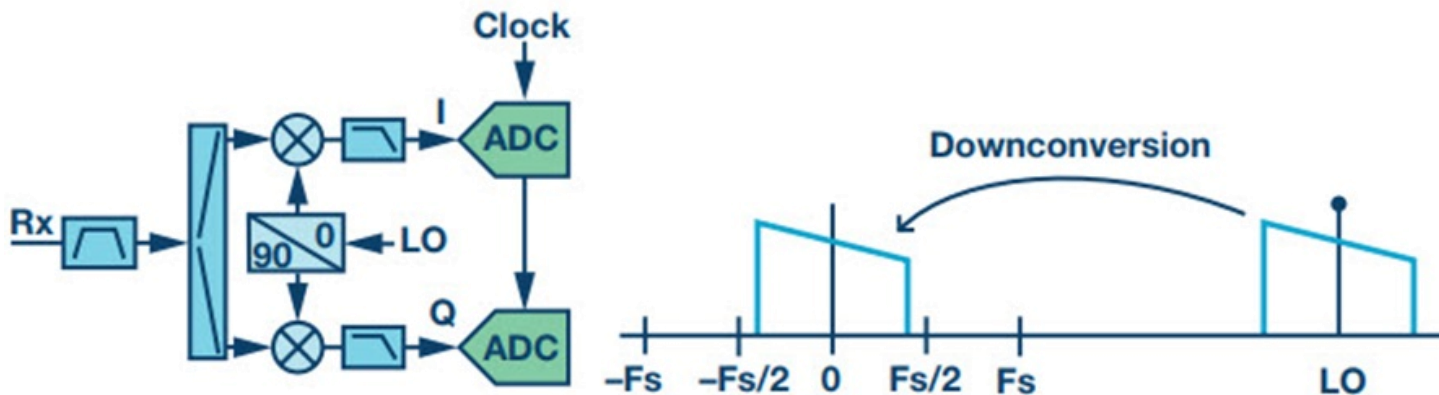


- Provides hardware abstraction layer
 - Allows sharing of infrastructure
 - Allows developers to focus on the solution
 - Allows application re-use
- Kernel drivers have low-level & low-latency access to hardware
 - MMIO
 - Interrupts
 - DMA
 - Memory
- IIO provides fast and efficient data transport
 - From device to application
 - From application to device
 - From device to network/storage



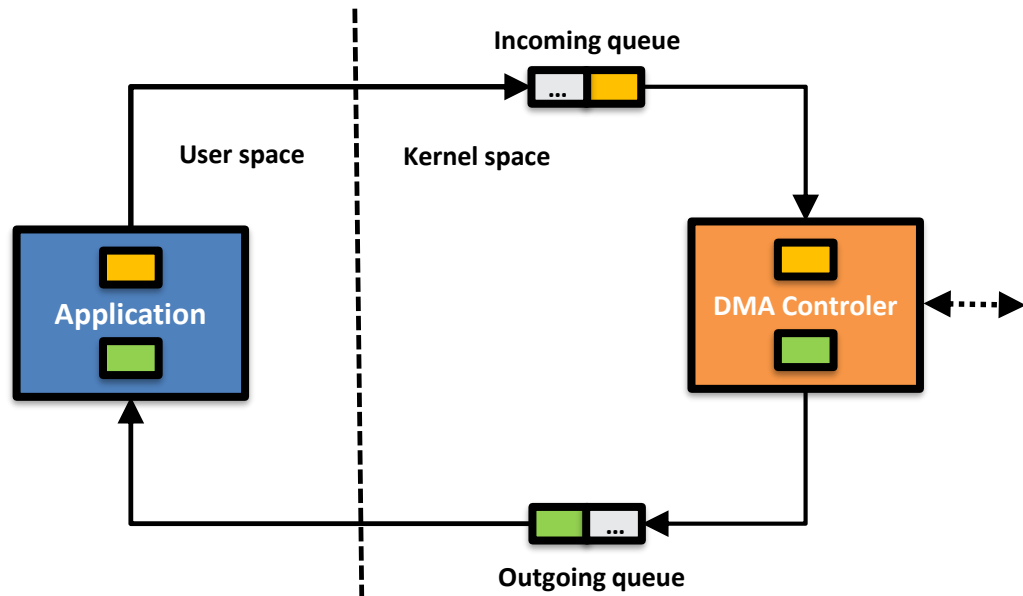
Fast and efficient data transport a SDR requirement ?

- Nyquist sampling 20 MHz of RF bandwidth e.g FM Radio from 88...108 MHz produces a constant data stream of 80 MBytes/s @ 16bits !
 - PlutoSDR max sample rate = 61.44 MSPS -> 245.76 MBytes/s
 - This is way to fast for USB2.0
 - **Efficient low latency, zero copy data transfers are required**



IIO – DMA buffer

- DMA is used to copy data from device to memory
- `mmap()` is used to make data available in the application
- Allows low overhead high-speed data capture
- Data is grouped into chunks (called DMA blocks) to manage ownership
 - Either application or driver/hardware owns a block
 - Samples per block are configurable
 - Number of blocks are configurable



Fast Boot optimizations

- Many good recipes – no one size fits all!
- Less is more
 - Slim down kernel size
 - Remove unnecessary kernel options
 - Slim down root file system
 - Use BusyBox
 - Lightweight init system
 - /etc/inittab
 - /etc/init.d/rcS
 - mdev
 - Use silent boot
 - Kernel cmd line: `quiet loglevel=4`
 - u-boot: `set stdout nulldev`
 - Use the initial RAM disk (initrd)
 - u-boot: `set bootdelay 0`



Many more things can
be done but this was enough
to get down to 3 seconds

Dealing with an embedded Linux system while users consider hotplug/removal

- Use ramfs (initramfs)
- **Restrict flash memory write access**
 - Allow Firmware Upgrades
 - Allow u-boot env storage
 - Latest firmware enables JFFS2 on **partition@qspi-nvmfs**
 - Persistent ssh keys and password
- Flash block protection on **partition@qspi-fsbl-uboot**
- Fail-Safe mechanism
 - Enter DFU mode if ulmage.FIT fails
 - U-boot use compiled in environment
 - Bad CRC
 - Button pressed
 - JFFS2 files have MD5
-

```
&qspi {
    status = "okay";
    is-dual = <0>;
    num-cs = <1>;
    primary_flash: ps7-qspi@0 {
        #address-cells = <1>;
        #size-cells = <1>;
        spi-tx-bus-width = <1>;
        spi-rx-bus-width = <4>;
        compatible = "n25q256a", "n25q512a", "jedec,spi-nor";
        reg = <0x0>;
        spi-max-frequency = <50000000>;
        partition@qspi-fsbl-uboot {
            label = "qspi-fsbl-uboot";
            reg = <0x0 0x100000>; /* 1M */
        };
        partition@qspi-uboot-env {
            label = "qspi-uboot-env";
            reg = <0x100000 0x20000>; /* 128k */
        };
        partition@qspi-nvmfs {
            label = "qspi-nvmfs";
            reg = <0x120000 0xE0000>; /* 1M */
        };
        partition@qspi-linux {
            label = "qspi-linux";
            reg = <0x200000 0x1E00000>; /* 30M */
        };
    };
};
```

} Write Protected

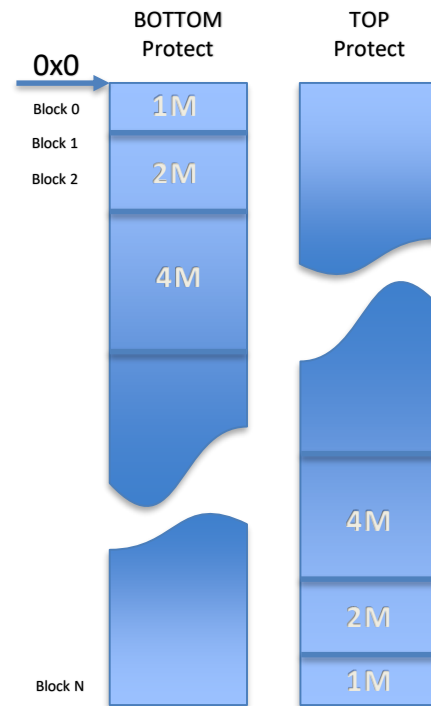
} Fail-Safe Mechanism

} Not Essential



Flash Block Protection (Locking)

- Block Protection feature
 - TOP/BOTTOM Protect
 - SoC SPI boot mode
 - Vendor specific
 - Nonvolatile control bits
 - **drivers/mtd/spi-nor** preserves these bits
 - Block Protection enabled in u-boot during factory programming



Reboot bootloader actions

- Kernel instruct bootloader to enter a nondefault boot mode after RESET
 - DFU mode
 - DFU RAM mode
 - Break in u-boot
 - Start system verbose
- Normally use fw_setenv in the kernel and check the variable in u-boot.
- On Pluto we want to restrict flash write access to the bare minimum.
 - Use general purpose R/W field in a system level control register that persists through all resets except a POR reset. (Not assigned or written by the BootROM or hardware)

```
#!/bin/sh

case "$1" in
    ram)
        cause=7
        ;;
    sf)
        cause=3
        ;;
    verbose)
        cause=6
        ;;
    break)
        cause=2
        ;;
    reset)
        cause=0
        ;;

    *)
        echo "Usage: $0 {ram|sf|reset|verbose|break}"
        echo "  sf      : Reboot and enter Serial Flash DFU mode"
        echo "  ram     : Reboot and enter RAM DFU mode"
        echo "  reset   : Reboot"
        echo "  verbose: Reboot and start serial console Verbose"
        echo "  break   : Reboot and HALT in u-boot"
        exit 1
esac

echo $cause > /sys/kernel/debug/zynq_rst/code && /sbin/reboot

exit 0
```



Kernel -> u-boot messaging

- Back in u-boot use itest command on the persistent register and execute appropriate boot mode.

```
clear_reset_cause=mw f8000008 df0d && mw f8000258 00400000 && mw f8000004 767b;

itest *f8000258 == 480003 && run clear_reset_cause && run dfu_sf; /* Enter DFU Mode */
itest *f8000258 == 480007 && run clear_reset_cause && run ramboot_verbose; /* Enter DFU-RAM Mode */
itest *f8000258 == 480006 && run clear_reset_cause && run qspiboot_verbose; /* Boot Verbose */
itest *f8000258 == 480002 && run clear_reset_cause && exit; /* Stop in u-boot */
```


Failsafe boot: u-boot env corrupted?

```
#ifndef CONFIG_MISC_INIT_R
#include <asm/gpio.h>
#include <environment.h>
int misc_init_r(void)
{
#define BUTTON_GPIO 14

    gpio_request(BUTTON_GPIO, "SWITCH");
    gpio_direction_input(BUTTON_GPIO);

    if (!gpio_get_value(BUTTON_GPIO))
        set_default_env("Button pressed: Using default environment\n");

    return 0;
}
#endif
```



Control and Interaction Concept

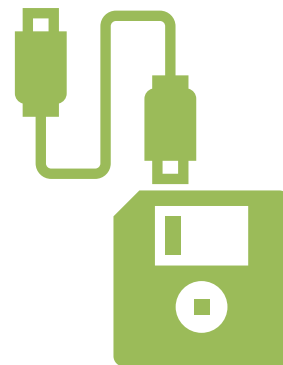
- How to control, configure and interact with a black box device that only has a hidden button, two LEDs and a USB OTG jack?
 - Avahi/Zeroconf
 - Webserver
 - LEDs for sign of life and state indication
 - Serial console, SSH (expert users)
 - Morse code with button? ☹
 - Special USB function with GUI? ☹
 - What else?

Video HERE

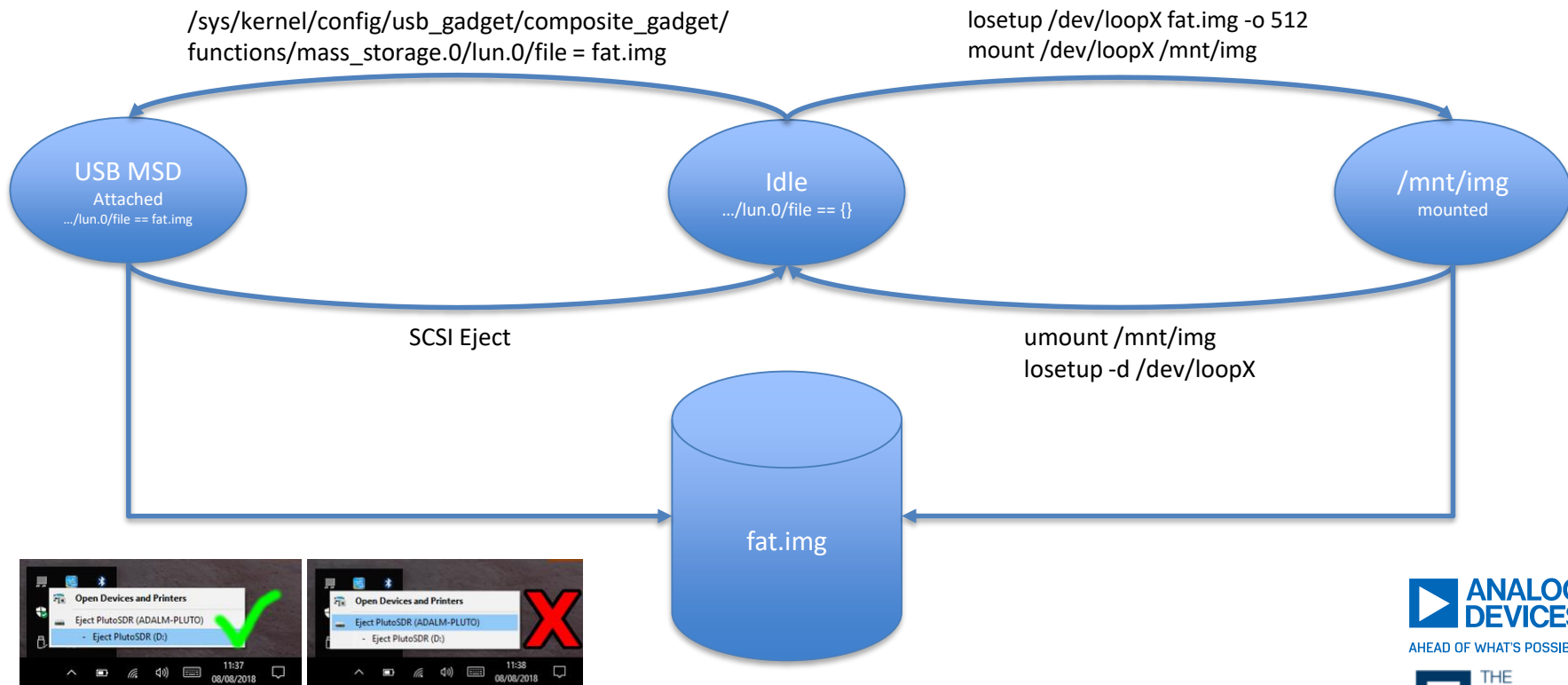
- Video HERE

USB Mass Storage Gadget (MSG)

- Allows your device to act as a USB mass storage device just like a pendrive.
- Requires a Backing storage
 - Block device or regular file.
 - Requires full sized file, with partition table and filesystem.
 - For cross platform use FAT partition and filesystem.
 - This file can't be accessed (loop mounted, etc.) while attached to the MSG, unless it's read only from both sides.



Time sharing the MSD backing storage (is possible)



<https://github.com/analogdevicesinc/buildroot/blob/pluto/board/pluto/update.sh>

LED class

- Allows control of LEDs from userspace
- LEDs are controlled via files in **/sys/class/leds/**
- Brightness will set the brightness of the LED
 - The minimum brightness is 0, and the maximum is 255 (max_brightness)
 - For GPIO controlled LEDs any value greater than 0 will turn the LED on
- LEDs class has an optional concept of LED triggers
 - Triggers are kernel based source of led events
 - heartbeat
 - timer
 - etc.

```
leds {
    compatible = "gpio-leds";
    led0 {
        label = "led0:green";
        gpios = <&gpio0 15 0>;
        linux,default-trigger = "heartbeat";
    };
};

flash_indication_on() {
    echo timer > /sys/class/leds/led0:green/trigger
    echo 40 > /sys/class/leds/led0:green/delay_off
    echo 40 > /sys/class/leds/led0:green/delay_on
}

flash_indication_off() {
    echo heartbeat > /sys/class/leds/led0:green/trigger
}
```

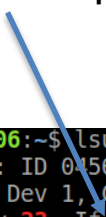


- gpio-keys driver translates GPIO interrupts into Linux input events
- BusyBox input event daemon can be used to invoke certain commands based on input events

```
gpio_keys {  
    compatible = "gpio-keys";  
    #address-cells = <1>;  
    #size-cells = <0>;  
  
    button {  
        interrupt-parent = <&gpio0>;  
        interrupts = <14 IRQ_TYPE_EDGE_FALLING>;  
        label = "Button";  
        linux,code = <BTN_MISC>;  
    };  
};  
  
#  
# /etc/input-event-daemon.conf  
#  
  
[Global]  
listen = /dev/input/event0  
  
[Keys]  
BTN_0          = ACTION=remove_all /lib/mdev/automounter.sh
```


Tips & Tricks: Multifunction USB gadget via configFS

- USB Gadget seen as a set of configurations
 - Each configuration has a number of interfaces also referred as functions.
 - Sometimes the order of interfaces matters!
 - Windows hosts requires RNDIS on Interface 0



```
michael@mhenner1-D06:~$ lsusb -s 23; lsusb -t | grep -E "23|Bus 01"
Bus 001 Device 023: ID 0456:b673 Analog Devices, Inc.
/: Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/16p, 480M
|   Port 8: Dev 23, If 0, Class=Communications, Driver=rndis_host, 480M
|   Port 8: Dev 23, If 1, Class=CDC Data, Driver=rndis_host, 480M
|   Port 8: Dev 23, If 2, Class=Mass Storage, Driver=usb-storage, 480M
|   Port 8: Dev 23, If 3, Class=Communications, Driver=cdc_acm, 480M
|   Port 8: Dev 23, If 4, Class=CDC Data, Driver=cdc_acm, 480M
|   Port 8: Dev 23, If 5, Class=Communications, Driver=, 480M
```

- https://www.kernel.org/doc/html/latest/usb/gadget_configs.html

Tips & Tricks: Gadget USB Serial Number

- In order to distinguish between different devices, the serial number stored in the device descriptor must be unique.
- Assigning unique and persistent IDs can be expensive
 - Use SPI flash Unique ID code (UID)
 - Read SPI NOR identification sequence.
 - Behind the JEDEC Manufacture and Device Identification often there is a 16 byte UID present.

Kernel messages

```
m25p80 spi1.0: SPI-NOR-UniqueID 104400b83991001807001100968f65adac  
m25p80 spi1.0: found n25q512a, expected n25q256a  
m25p80 spi1.0: n25q512a (65536 Kbytes)
```

USB Gadget ConfigFS

```
serial=`dmesg | grep SPI-NOR-UniqueID`  
serial=${serial#*SPI-NOR-UniqueID }
```

```
echo 0x0456 > $GADGET/idVendor  
echo 0xb673 > $GADGET/idProduct
```

```
mkdir -p $GADGET/strings/0x409  
echo "Analog Devices Inc." > $GADGET/strings/0x409/manufacturere  
echo "PlutoSDR (ADALM-PLUTO)" > $GADGET/strings/0x409/product  
echo $serial > $GADGET/strings/0x409/serialnumber
```



USB OTG HOST

- The Pluto will automount any USB mass storage device such as thumb drive or Hard Drives (ext4, msdos, vfat)
- Kernel hotplug:
 - `echo /sbin/mdev >/proc/sys/kernel/hotplug`
 - `sd[a-z][0-9] root:root 660 */lib/mdev/automounter.sh`
- The automounter will then look for some special file names:
 - `runme[0-9].sh` which it will run as a shell script
 - `runme[0-9]` which it will run as a binary file.
 - See: `/lib/mdev/automounter.sh`
- Pressing the button while a mass storage device is mounted will safely unmount it.
 - `/etc/input-event-daemon.conf`
- Switching from the slim down PlutoSDR buildroot rootfs to a full-blown ARM HF ubuntu root filesystem is easy
 - Using busybox `switch_root` command
 - chroot into a new filesystem and exec a new init process out of the new filesystem



1. Provide power to right USB
2. Plug OTG+USB drive into left USB port
3. Left LED will become solid
4. Script runs



Managing HW revisions and boot configurations

- There are always HW revisions
 - Can we handle those transparent to the user?
 - Single firmware file who rules them all?
 - Image integrity protection?
 - Fail-safe?

ulImage.FIT

<https://gitlab.denx.de/u-boot/u-boot/-/tree/master/doc/ulImage.FIT>



Flattened Image Trees (FIT Images)

- Multi-component image format
- Support for multiple configurations
- Support for FDT overlays
- Hashing/Signatures
- Verified boot
 - not needed here
- Device tree structure

```
bootm [<addr1>]#<conf>[#<extra-conf[#...]]  
  
echo Booting silently && set stdout nulldev;  
  
bootm ${fit_load_address}#${fit_config} ||  
    set stdout serial@e0001000;echo BOOT failed entering DFU mode ... &&  
    sf protect lock 0 100000 && run dfu_sf;
```

<https://gitlab.denx.de/u-boot/u-boot/-/tree/master/doc/ulmage.FIT>



Pluto.its (source image file)

```
/dts-v1/;

/ {
    description = "Configuration to load fpga before Kernel";
    magic = "ITB PlutoSDR (ADALM-PLUTO)";
    #address-cells = <1>;
    images {
        fdt@1 {
            description = "zynq-pluto-sdr";
            data = /incbin/("../build/zynq-pluto-sdr.dtb");
            type = "flat_dt";
            arch = "arm";
            compression = "none";
        };
        fdt@2 {
            description = "zynq-pluto-sdr-revb";
            data = /incbin/("../build/zynq-pluto-sdr-revb.dtb");
            type = "flat_dt";
            arch = "arm";
            compression = "none";
        };
        fdt@3 {
            description = "zynq-pluto-sdr-revc";
            data = /incbin/("../build/zynq-pluto-sdr-revc.dtb");
            type = "flat_dt";
            arch = "arm";
            compression = "none";
        };
        fpga@1 {
            description = "FPGA";
            data = /incbin/("../build/system_top.bit");
            type = "fpga";
            arch = "arm";
            compression = "none";
            load = <0xF000000>;
            hash@1 {
                algo = "md5";
            };
        };
    };
};
```

<https://github.com/analogdevicesinc/plutosdr-fw/blob/master/scripts/pluto.its>

```
linux_kernel@1 {
    description = "Linux";
    data = /incbin/("../build/zImage");
    type = "kernel";
    arch = "arm";
    os = "linux";
    compression = "none";
    load = <0x8000>;
    entry = <0x8000>;
    hash@1 {
        algo = "md5";
    };
};

ramdisk@1 {
    description = "Ramdisk";
    data = /incbin/("../build/rootfs.cpio.gz");
    type = "ramdisk";
    arch = "arm";
    os = "linux";
    compression = "gzip";
    hash@1 {
        algo = "md5";
    };
};

};

configurations {
    default = "config@0";
    config@0 {
        description = "Linux with fpga RevA";
        fdt = "fdt@1";
        kernel = "linux_kernel@1";
        ramdisk = "ramdisk@1";
        fpga = "fpga@1";
    };
};

/* all below is currently RevB ! */

config@1 {
    description = "Linux with fpga RevB";
    fdt = "fdt@2";
    kernel = "linux_kernel@1";
    ramdisk = "ramdisk@1";
    fpga = "fpga@1";
};
```



Managing Configurations / HW Revisions

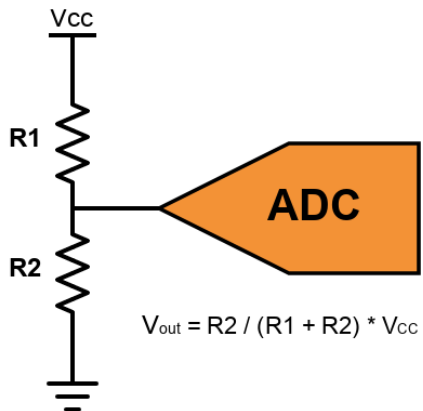
```
static int do_pluto_hw_version(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
    int val, ret = 0, i;
    char buf[16];

    if (pluto_revA) {
        setenv("PlutoRevA", "1");
    } else {
        val = do_xadc(cmdtp, flag, 1, argv);
        setenv("PlutoRevA", "");
        for (i = 100; i <= 1000; i += 100) {
            if ((val >= (i - 50)) && (val < (i + 50))) {
                ret = i / 100;
                break;
            }
        }
    }

    snprintf(buf, sizeof(buf), "config@%d", ret);
    setenv("fit_config", buf);

    return ret;
}

U_BOOT_CMD(
    pluto_hwref, CONFIG_SYS_MAXARGS, 1, do_pluto_hw_version,
    "determine pluto hw revision",
    ""
);
```



- Custom pluto_hwref u-boot command
 - Read board revision set by voltage divider using a SoC ADC.
 - Populate env variable with the corresponding config variable. (config@X)
 - X = {0,1,2, ..., 10}
- `bootm ${fit_load_address}#${fit_config}`
- `bootm 0x2080000#config@X`



Is it working?

- Yes, we have shipped nearly 40,000 devices since release in 2017
- Many users in Education and Academia, but also in the HAM Radio community
- Adopted in many Open Source projects by the community
- Customer feedback is always very positive – “it just works”
- Making things work on VM (via RNDIS) is super easy
- Libiio is very stable, and cross platform



Links and Pointers

Analog Devices Active Learning Modules and University Engagements

- [1] <https://www.analog.com/en/education/university-engagement.html>
- [2] <https://www.analog.com/en/education/courses-and-tutorials/active-learning-module.html>

Wiki Documentation

- [3] <https://wiki.analog.com/university/tools/pluto>

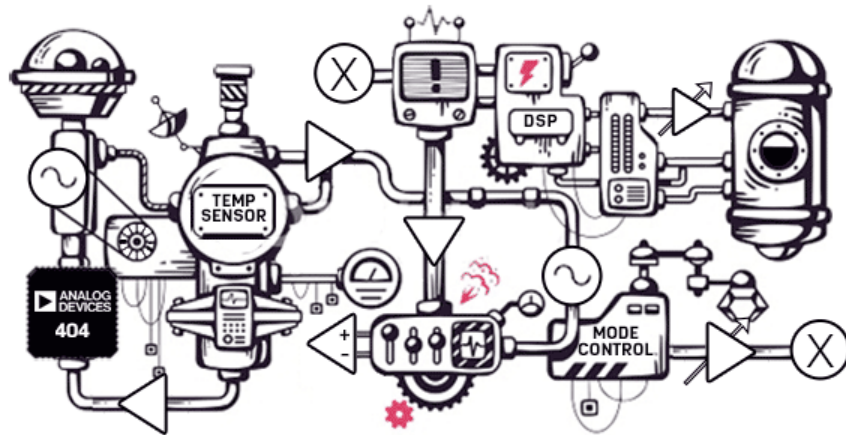
The source code:

- [4] <https://github.com/analogdevicesinc/plutosdr-fw>

Free books on SDR

- [5] <http://www.analog.com/sdrforengineers>
- [6] <http://pysdr.org/>





Ahhh, technology. We can't find that page.

Thanks

Q & A



Embedded Linux Conference Europe